A Gentle Introduction to CP - MII 771

Constraint Programming

Dr. Ricardo Soto

Constraint Satisfaction involves various solving approaches, mainly based on mathematics and on artificial intelligence techniques. In this paper we give an overview of these approaches. We introduce first the formal definition of a CSP and then we present the basis of constraint programming techniques, starting from the basic form of systematic search to more complex resolution mechanisms. Next, we present some techniques from the mathematical field followed by a summary of metaheuristics as a parallel approach devoted to tackle optimization problems.

1 Formal definition of a CSP

A Constraint Satisfaction Problem $\mathcal{P}$ is defined by a triple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where:

- $\mathcal{X}$ is a n-tuple of variables $\mathcal{X} = \langle x_1, x_2, ..., x_n \rangle$,
- $\mathcal{D}$ is a corresponding n-tuple of domains $\mathcal{D} = \langle D_1, D_2, ..., D_n \rangle$ such that $x_i \in D_i$, and D_i is a set of values, for $i = 1, ..., n$.
- $\mathcal{C}$ is a t-tuple of constraints $\mathcal{C} = \langle C_1, C_2, ..., C_t \rangle$.

CSPs can also be defined by means of graphs, where each node is represented by a variable, labelled with its domain, and the edges represent the constraints. For instance the CSP $\mathcal{P} = \langle \langle x, y, w, z \rangle, \langle D_x \in \{0, 1\}, D_y \in \{1, 2\}, D_w \in \{2, 3\}, D_z \in \{3, 4\} \rangle, \langle C_1 = \{x \leq y\}, C_2 = \{y \leq w\}, C_3 = \{w \leq z\}, C_4 = \{y + 2 \leq z\} \rangle \rangle$ is represented as a graph in Figure 1.

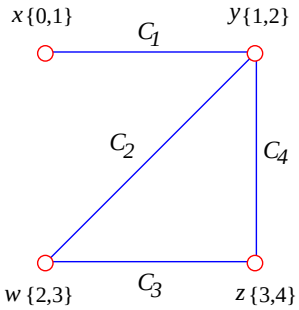

Figure 1: A CSP represented by a graph.

The classic CSP attempt to find the first solution that satisfies the whole set of constraints. This classic definition can be extended to represent optimization problems, problems whose focus is searching the best solution(s) for a given criterion. Under the constraint satisfaction framework, optimization problems are called constraint optimization problems (COP), which can be defined by a 4-tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{O} \rangle$ where O is an objective function corresponding to the given criterion to be minimized or maximized.

The general CSP can also be specialized in other categories e.g., depending on the values of the domains. As an example, a finite domain CSP stands for a CSP involving only integer values, a numerical CSP refers to CSP on reals. Another examples are the Boolean CSPs and the symbolic CSPs, the latter considers non-numeric domains. For an extended presentation of CSP categories please refer to [6, 14]

2 Constraint Programming Techniques

Constraint satisfaction, in its basic form, involves finding values for each one of the variables of the problem. One of the simplest methods to perform this task is by means of the Generate and Test (GT) algorithm. This method consist in generating a potential solution and checking it whether satisfies all the constraints. This process is done systematically taking the potential solution from the set corresponding to all the possible combinations among the values of the variables (the cartesian product of all the variable's domains).

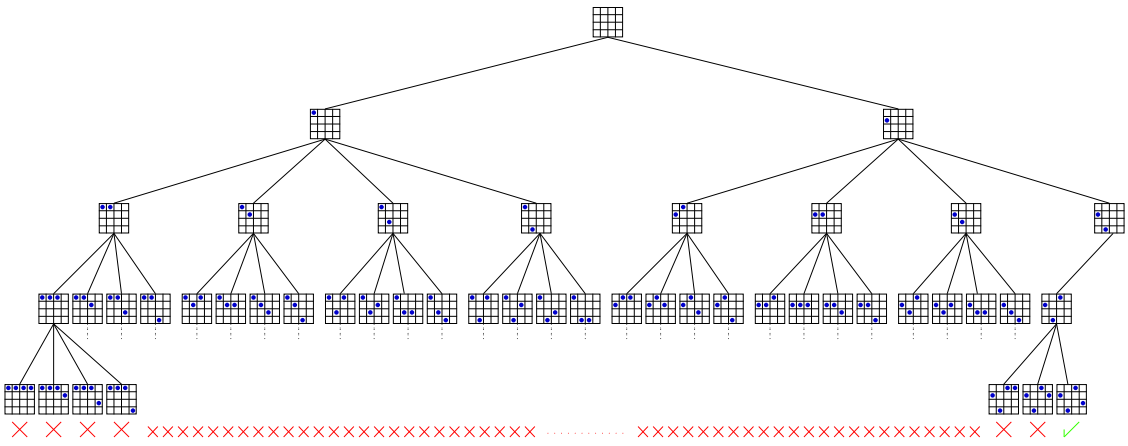

Figure 2: Solving the 4-queens problem using GT.

Let us illustrate the GT process by means of the 4-queens problem. Figure 2 depicts an extract of the process done by the GT algorithm to reach a result for this problem. Regarding the figure is easy to see that the GT procedure is very naive, constraints are tested always with all the variables instantiated (at the lower level of the tree), so if a partial solution violates a constraint it cannot be detected as soon as the variables involved in the partial solution have been instantiated. This led to a generation of several wrong assignments that could be detected earlier. This approach is simple to implement, however the searching cost is likely to be very high, in fact it is proportional to the number of candidate solutions, which, in many problems tends to grow as quickly as the size of the problem increases. As a consequence, the use of GT is feasible only to problems of reduced size.

2.1 Backtracking

Backtracking (BT) is another approach for performing systematic search. In the BT method potential solutions are generated incrementally by repeatedly choosing a value for another variable and as soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. So, if a partial solution violates any of the constraints, the algorithm returns to the most recently instantiated variable that still has alternatives available, eliminating as consequence the conflicting subspace.

Figure 3 depicts the search process performed by the BT procedure on the 4-queens problem. Here, we can see that BT is able to detect failures as soon as two variables are instantiated (at the middle level of the tree), much earlier than in the GT approach. Despite of this, the BT approach is not able to detect failures before assigning the values to all the variables involved in a conflicting constraint. This problem can be handled by applying local consistencies.

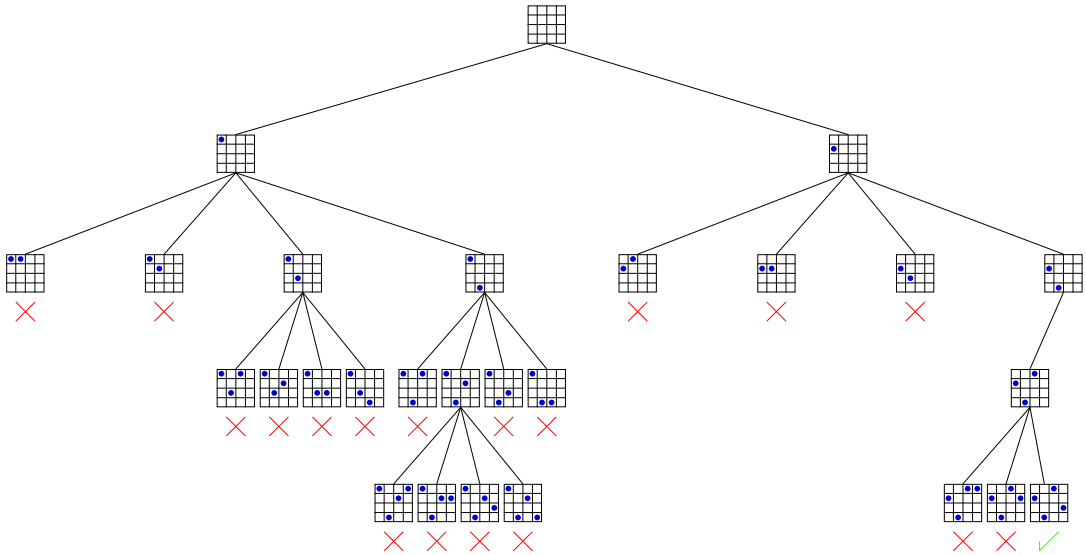

Figure 3: Solving the 4-queens problem using BT.

2.2 Consistency Techniques

Consistency Techniques are filtering mechanisms that aim to improve the performance of the search process by attempting to reduce the search space. This reduction is possible by means of a process called constraint propagation where a determined level of consistency among subsets of variables is enforced. This level of consistency is technically called local consistency and there exists various types, for instance node consistency, arc consistency and path-consistency.

2.2.1 Node Consistency

Node consistency is the simplest of the local consistencies and it dealt with unary constraints. A CSP is node consistency if for every value in the current domain of a variable x, each unary constraint on x is satisfied. If the domain of the variable x contains values that does not satisfy the unary constraint involving V, the node inconsistency can be eliminated by simply removing the values from the domain of x that do not satisfy the unary constraint on V.

For instance the CSP composed by the constraint $x > 1$ over the domain $D_x = [1..4]$ of x is not node consistent since the value 1 does not satisfy the constraint.

2.2.2 Arc Consistency

Arc consistency dealt with binary constraints. A binary constraint involving a variable x and a variable y is arc consistency if every value in the domain of x and y participates in a solution. A CSP is arc consistency if all its binary constraints are arc consistent.

For instance, the CSP shown in Figure 4, which consists of one constraint $x < y$ over the domains $D_x = [3..6]$ of x and $D_y = [4..6]$ of y is not arc consistent since the value 6 in D_x does not participate in any solution.

Let us remark that arc consistency does not imply consistency and consistency does not imply arc consistency either. Take for instance the CSP $\mathcal{P} = \langle\langle x, y\rangle, \langle D_x \in \{0, 1\}, D_y \in \{0, 1\}\rangle, \langle x =$

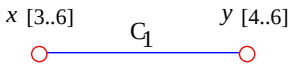

Figure 4: A non arc consistent CSP.

$y, x \neq y\rangle\rangle$ depicted in Figure 5, which is arc consistent but inconsistent; and the CSP $\mathcal{P}\langle\langle x, y\rangle, \langle D_x \in \{0, 1\}, D_y = \{0\}\rangle, \langle x = y\rangle\rangle$ depicted in Figure 6, which is consistent but not arc consistent.

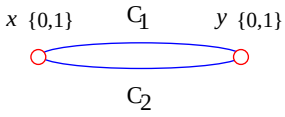

Figure 5: A arc consistent, but inconsistent CSP.

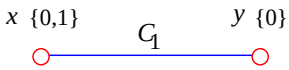

Figure 6: A consistent, but non arc consistent CSP.

Constraint propagation can make the problem arc consistent by deleting those values for which there is no corresponding value in the other domain such that the binary constraints are satisfied. This process might have to consider a given pair of variables more than once since removing values from the domain of a variable may cause other variables to become no longer arc consistent with it. These concerns have led the design of many algorithms to enforce arc consistency, for instance AC-2 [16], AC-3 [8] and AC-4 [10]. Currently arc consistency is the most used local consistency in constraint satisfaction systems.

2.2.3 Hyper Arc Consistency

The notion of arc consistency can be generalized in the Hyper arc consistency, so a constraint is hyper-arc consistent if for every involved domain, each element of it participates in a solution.

For instance the boolean CSP $\mathcal{P} = \langle\langle x, y, z\rangle, \langle D_x \in \{0, 1\}, D_y \in \{0, 1\}, D_z \in \{0, 1\}\rangle, \langle x \vee y = z\rangle\rangle$ shown in Figure 7 is hyper arc consistent since each element of every domain participates in a solution. In contrast, the CSP $\mathcal{P} = \langle\langle x, y, z\rangle, \langle D_x \in \{1\}, D_y \in \{1\}, D_z \in \{0, 1\}\rangle, \langle x \Rightarrow y = z\rangle\rangle$ shown in Figure 8 is not hyper arc consistent since the value 0 of D_z does not participate in any solution.

There also exists stronger ways of applying filtering, which may eliminate a bigger number of conflictive values on domains, but at higher cost in terms of computations. Some examples are the path consistency and the k-consistency. For a detailed presentation of all the consistency techniques please refer to [6].

2.3 Constraint Satisfaction Algorithms

Systematic search algorithms and consistency techniques are rarely used alone. The most common approach is to combine the BT algorithm with a consistency technique. In this way, the algorithm attempts to reduce the search space by applying a consistency technique after a value is assigned to a variable. Depending on the degree of the consistency technique used, there exists various constraint satisfaction algorithms.

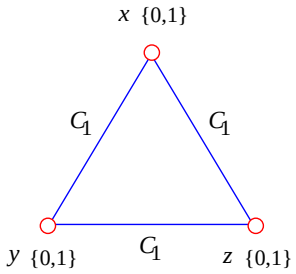

Figure 7: A hyper arc consistent CSP.

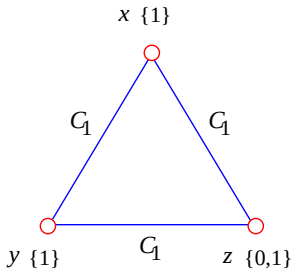

Figure 8: A non hyper arc consistent CSP.

The backtracking approach performs in a certain sense a kind of a consistency technique since checks the validity of constraints considering partial instantiations, this can be seen as to apply a fraction of arc consistency among already instantiated variables. In this approach the consistency is applied to the instantiated variables so it is not possible to reduce the search space before this instantiation.

2.3.1 Forward Checking

The forward checking (FC) approach tackle this concern. Forward checking is able to prevent future conflicts by performing arc consistency to the not yet instantiated variables. This is done by removing temporarily the values of the variables that will further cause conflict with the current variable assignment. Hence, the algorithm knows immediately that the current partial solution is inconsistent and consequently the search space can be pruned earlier than using simple backtracking.

Figure 9 illustrates this process, values from domains are removed since the second level of the tree. Once a queen is stated, their future conflictive values are temporarily removed, for instance the queens stated at the position (1,1) removes all values corresponding to the first row and the NW-SE diagonal (indicated by the blue arrows). Then, at the left subtree, the second queens is placed on the position (3,2) which is immediately set as inconsistent since it does not leave available place for the third queen. The propagation follows for every queen stated on the chessboard allowing to avoid most of wrong instantiations done by the BT approach.

2.3.2 Maintaining Arc Consistency

The process done by forward checking can be even more ambitious, for instance if we also check the conflicts between future variables (in addition to the test between the current and the future

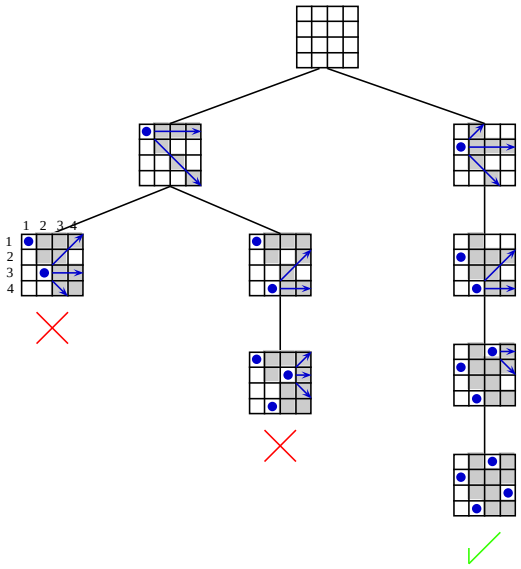

Figure 9: Solving the 4-queens problem using FC.

variables). This approach is called Full Look Ahead or Maintaining Arc Consistency (MAC).

Figure 10 illustrates this process, where we can see that the MAC algorithm is able to prune the search space earlier than the forward checking, but doing a bigger work on each variable assignment. For instance, when the first queen is placed at the position (1,1) the conflicts between the current position and the future positions are removed (indicated with the blue arrows). After that, the algorithm checks the conflicts among the future variables starting with the first available position on the second column, the cell (3,2). The algorithm find out that the position (3,2) is inconsistent since does not leave available place for the third queen (orange arrows), thus the position (3,2) is removed. The algorithm follows with the cell (4,2), the next available position on the second column. This placement (green arrows) leaves the (2,3) cell as the unique available position on the third column which is then set as inconsistent since it does not leave available place for the fourth queen (red arrows). The process follows until the result is reached on the right subtree.

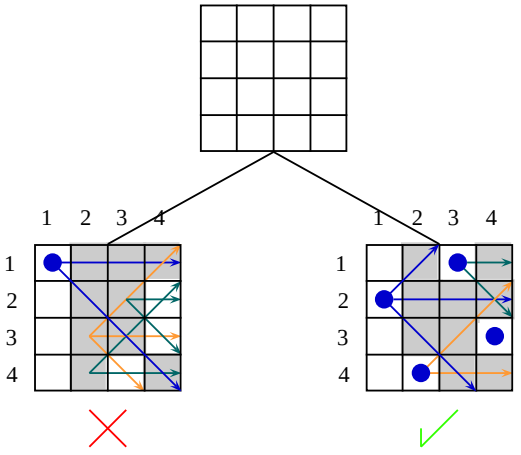

Figure 10: Solving the 4-queens problem using MAC.

2.4 Value and Variable Ordering Heuristics

In the previous examples we have started the search process by selecting a variable (the first queen) and then assigning it a value of its domain (the first value), the order in which this choice is done are referred to as the variable and the value ordering. Several experiments have demonstrated that a correct ordering decision for variables as well as for values can be crucial to perform an effective solving process [6].

The ordering of the values and variables can be either static or dynamic. In the static ordering the order is specified before the search begins, and in the dynamic ordering the selection of the next variable depends on the information of the current state of the search. Dynamic ordering cannot be applied in any search algorithm, for instance using simple backtracking there is no extra information during the search able to change the initial ordering defined. On the contrary, using forward checking, the domains of further variables may change during an instantiation giving additional information which may be used to choose the next variable.

2.4.1 Variable Ordering

There exists several heuristic for selecting the variable ordering. For instance the most common are:

- Select the variable with the smallest domain:
  This choice is motivated by the assumption that a success can be achieved by trying first the variables that have a bigger chance to fail, in this case, the values with a smaller number of available alternatives.
- Select the most constrained variable:
  This choice can be justified by the fact that the instantiation of such a variable should lead to a bigger tree pruning through the constraint propagation.

2.4.2 Value Ordering

The order in which the values are selected can also have a considerable impact. For example, if the right value is chosen on the first try for each variable, a solution can be found without performing backtracks. However, if the CSP is inconsistent or the whole set of solutions is required, the value ordering is irrelevant. The literature presents some obvious ways to perform this selection which, depending on the problem, may lead to a more efficient constraint propagation [6]:

- Selects the smallest value.
- Selects the median value.
- Selects the maximal value.

There also exist more complex value ordering heuristics which are in general either based on estimating the number of solutions or estimating the probability of a solution [14].

3 Global Optimization

When solving problems under constraints we may be interested in search the best solution instead of searching the first one. In this context, much more processing must be done to achieve the goal, as it the search process must check several solutions. In the next paragraphs we give an overview of the major approaches for global optimization.

3.1 Mathematical Programming

Mathematical programming is a branch of applied mathematics and numerical analysis devoted to tackle optimization problems. Techniques to deal with mathematical programming depend both on the nature of the objective function and on the constraint set. Some examples are integer programming, linear programming, and non-linear programming [11, 2, 15].

3.1.1 Linear Programming

Linear Programming studies the case in which the objective function is linear and the set of constraints is specified only using linear equalities and inequalities. In geometric terms, a feasible region is defined by the linear constraints and the objective function in the form of a convex polytope where the optimum, or optimums, are placed on their vertex or facets. One of the methods to solve these problem is called the simplex algorithm which move along the facets of the polytope, from point to point, until the optimum is reached.

3.1.2 Integer Programming

If the unknown variables are all required to be integers, then the problem is called an integer programming (IP) or integer linear programming (ILP) problem. If only some of the unknown variables are required to be integers, then the problem is called a mixed integer programming (MIP) problem. Some algorithms to solve ILP programs are the Cutting-plane method, the Branch and Bound Brand and Cut algorithms.

3.1.3 Cutting-plane method

The cutting-plane method [12] is used to find integer solutions of a linear program. The basic idea of this method is as follows: First, the ILP problem is relaxed to an LP problem and then solved by an LP solver. If the optimum found is a non-integer solution, a new restriction is added that cuts off the non-integer solution but does not cut off any integer points of the feasible region. This is repeated until an optimal integer solution is found.

3.1.4 Branch and Bound

Branch and Bound (BB) is a general algorithm for finding optimal solutions. It can be view as a modification of the backtracking search where the value of an objective function is taken into account. The search is done systematically as in backtracking but including a global variable that maintains the current best value of the objective function, so instantiations are evaluated in terms of the global variable. If the current instantiation is "better" the global variable is updated otherwise it is discarded. Branch and bound can also be combined with the cutting-plane method to generate the branch and cut algorithm.

3.1.5 Non-linear programming

Nonlinear programming studies the general case in which the objective or constraints or both contain non-linear parts. If the objective function is concave, or convex and the constraint set is convex, then the program is called convex and general methods from convex optimization can be used (e.g. Ellipsoid method [11], Subgradient method [13], Cutting-plane methods). If the problem is non-convex, it can be solved by using branch and bound techniques.

3.2 Metaheuristics

For certain problems to find the best solution may very expensive in terms of computation time. So, often it is preferable to find a good solution in a fixed amount of time instead of finding the best one. Metaheuristics are based on this principle and for many problems are more effective that to carry out an exhaustive search. Metaheuristic can be seen as a set of concepts to define heuristics methods which considering a limited set of modifications can be applied to solve different optimization problems.

3.2.1 Simulated Annealing

Simulated Annealing (SA) [5] is based on this idea. The method consider that each candidate of the search space is analogous to a state of some physical system, and the function to be minimized represents the internal energy of the system in that state. The goal is to bring the system from an initial state to another state with the minimum possible energy.

The SA algorithm carry out transitions between one state to another by using a probability formula which depends on the function values (the energies) of the corresponding states and on a global parameter called the temperature which is gradually decreased during the process. This probability is normally chosen so that the system ultimately tends to move to states of lower energy. The algorithm follow the process repeatedly until the system reaches a state that is good enough for the application.

3.2.2 Genetic algorithms

Genetic algorithms (GA) [3] is another technique for solving optimization problems. GA belongs to the so-called evolutionary algorithms that use techniques inspired by evolutionary biology such as mutation, selection, and crossover.

Genetic algorithms consists in simulating the evolution of a initial population (which represent candidate solutions) toward a better population (better solutions). Traditionally, solutions are represented in binary, but other encodings are also possible. The evolution usually starts from a population composed of randomly generated individuals. In each generation, the quality (technically called fitness) of every individual in the population is evaluated to select a subgroup which is then modified (recombined or mutated) to create a new population. This new population will be used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

3.2.3 Ant colony optimization

Ant colony optimization [7] is a probabilistic technique for solving optimization problems inspired by the behavior of ants finding food. The behavior is interesting since ants initially begin searching at random, but once food is found they return to the colony leaving a pheromone trail, so next ants can be follow the pheronome to reach the food quickly, and if they eventually find food they will reinforce the amount of pheromone arising the attractiveness of the path. Over the time, the pheromone trail starts to evaporate, thus reducing its attractive strength. While more time is taken to travel over the path and back again, more time the pheromones have to evaporate. Hence, shorter paths are normally more attractive.

Thus, when one ant finds a short path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads all the ants following a single

path. The idea of the ant colony algorithm is to roughly imitate this behavior. More specifically, an ant is a simple computational agent, which iteratively constructs a solution for the instance to solve. Partial problem solutions are seen as states and the algorithm iterates to move ants from a state to another one, this transition is carried out by using a probability distribution that depends on the attractiveness of the move (computed by a given heuristic) and on the pheromone level of the path.

3.2.4 Particle swarm optimization

Particle swarm optimization (PSO) [4] is another problem solving technique belonging to the class of evolutionary algorithm. The technique is inspired by the social behavior of bird flocking or fish schooling which to find food learn from the current scenario.

In the PSO approach a population of individuals defined as random guesses at the problem solutions is initialized. These individuals are the candidates solutions (called particles). The proposed solutions are evaluated by means of a fitness function. A communication structure is also defined, assigning neighbors for each individual to interact with. An iterative process to improve these candidate solutions is set in motion. The fitness of the particles is iteratively evaluated and the location where they had their best success is remembered. Each particle makes this information available to their neighbors, so particles are able to see where their neighbors have had success. Movements through the search space are guided by these successes, with the population usually converging, by the end of a trial, on a problem solution.

3.2.5 Local Search & Hill Climbing

Local search [1] is another metaheuristic for solving optimization problems. Local search algorithms starts from a candidate solution and then iteratively moves to a neighbor solution. Typically, every candidate solution has more than one neighbor solution; the choice of which one to move to is taken using only information about the solutions in the neighborhood of the current one, hence the name local search. When the choice of the neighbor solution is done by taking the one locally maximizing the criterion, the metaheuristic takes the name hill climbing. Termination of local search can be defined by a time bound or when the best solution found by the algorithm has not been improved in a given number of steps.

3.2.6 Variable Neighborhood Search

The basic idea of Variable Neighborhood Search [9] is to systematically change the neighborhood of the current solution which is searched in order to explore an increasingly large region of the solution space. The procedure can be described as follows. In the initialization stage a set of different neighborhood structures is selected, an initial solution determined and a suitable stopping criteria chosen. In the next stage, starting with the first neighborhood structure a random neighbor of the initial solution is determined, and a local search heuristic is applied to find a solution. If this solution represents an improvement to the initial solution, the move is accepted and the same process is repeated starting with this new solution. Otherwise, the local optimum is abandoned, and a different usually larger neighborhood is selected for starting the process again.

References

[1] E. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization*. John Wiley and Sons, NY, USA, 1997.

[2] G. Dantzig. *Linear Programming and Extensions,*. Princeton University Press, 1963.

[3] D. Goldberg, D. Kalyanmoy, and J.Horn. Genetic algorithms. In *in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[4] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[6] K.R. Apt. *Principles of Constraint Programming*. Cambridge Press, 2003.

[7] M., V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:29–41, 1996.

[8] A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[9] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & OR*, 24(11):1097–1100, 1997.

[10] R. Mohr and T. Henderson. Arc and path consistency revisited. *Artificial Intelligence.*, 28(2):225–233, 1986.

[11] S. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, 1996.

[12] G. Nemhauser and L.Wolsey. *Integer and Combinatorial Optimization*. John and Wiley and Sons, 1999.

[13] R. Rockafellar. *Convex analysis*. Princeton University Press, 1854.

[14] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[15] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1998.

[16] D. Waltz. Understanding Line Drawings of Scenes with Shadows. In *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, 1975.